Modules and Packages in Python

Modules

- **Modules** - a module is a piece of software that has a specific functionality. Each module is a **different file**, which can be edited separately.
- **Modules** provide a means of **collecting sets of Functions together** so that they can be **used by** any **number of programs.**

Packages

- Packages – **sets of modules that are grouped together**, usually because their modules provide **related functionality** or because they **depend on each other**.

Modules

- **programs are designed to be run**, whereas **modules are designed to be imported** and **used by programs**.
- Several syntaxes can be used when importing. For example:
- import *importable*
- import *importable1, importable2, ..., importableN*
- import *importable as preferred_name*

- make the **imported objects** (variables, functions, data types, or modules) **directly accessible.**
- **from …import** syntax to import lots of objects.
- **Here are some other import syntaxes:**
- from *importable import object as preferred_name*
- from *importable import object1, object2, ..., objectN*
- from *importable import (object1, object2, object3, object4, object5, object6, ..., objectN)*
- from *importable import **

Python import statement

- We can **import a module** using the **import statement** and **access** the **definitions** inside it using **the dot operator**.
- import math
- print("The value of pi is", math.pi)

Import with renaming

- We can import a module by renaming it as follows:
- # import module by renaming it
- import math as m
- print("The value of pi is", m.pi)

Python from...import statement

- We can **import specific names** from a module **without importing the module as a whole**. Here is an example.

- # import only pi from math module
- from math import pi
- print("The value of pi is", pi)

Import all names

- We can import all names(definitions) from a module using the following construct:

- from math import *
- print("The value of pi is", pi)

The dir() built-in function

- We can use the dir() function to find out names that are defined inside a module.
- we have defined a function add() in the module example that we had in the beginning.
- dir(example)
- ['__builtins__', '__cached__', '__doc__', '__file__', '__initializing__', '__loader__', '__name__', '__package__', 'add']

- a sorted list of names (along with add).
- All other names that begin with an **underscore** are **default Python attributes associated with the module** (not-user-defined).

Let us create a module

- Type the following and save it as **example.py**.
- # Python Module example
- def add(a, b):
  - """This program adds two numbers and return the result"""
  - result = a + b
  - return result

How to import modules in Python?

```
import example

example.add(4,5.5)
9.5  # Answer
```

Variables in Module

- The module can **contain functions**, as already described, but also **variables** of all types (arrays, dictionaries, objects etc):
- Save this code in the file mymodule.py

-
  ```
  person1 = {
    "name": "John",
    "age": 36,
    "country": "Norway"
  }
  ```

- Import the module named mymodule, and access the person1 dictionary:
- import mymodule

  ```
  a = mymodule.person1["age"]
  print(a)
  ```
- Run Example → 36
-

Built-in Modules

- Import and use the platform module:

- import platform

```
x = platform.system()
print(x)
```

Import From Module

- The module named **mymodule** has one function and one dictionary:

-
  ```
  def greeting(name):
    print("Hello, " + name)

  person1 = {
    "name": "John",
    "age": 36,
    "country": "Norway"
  }
  ```
- Example
- Import only the person1 dictionary from the module:

-
  ```
  from mymodule import person1

  print (person1["age"])
  ```

-
  ```
  def greeting(name):
    print("Hello, " + name)

  person1 = {
    "name": "John",
    "age": 36,
    "country": "Norway"
  }
  ```
- Example
- Import all objecs from the module:

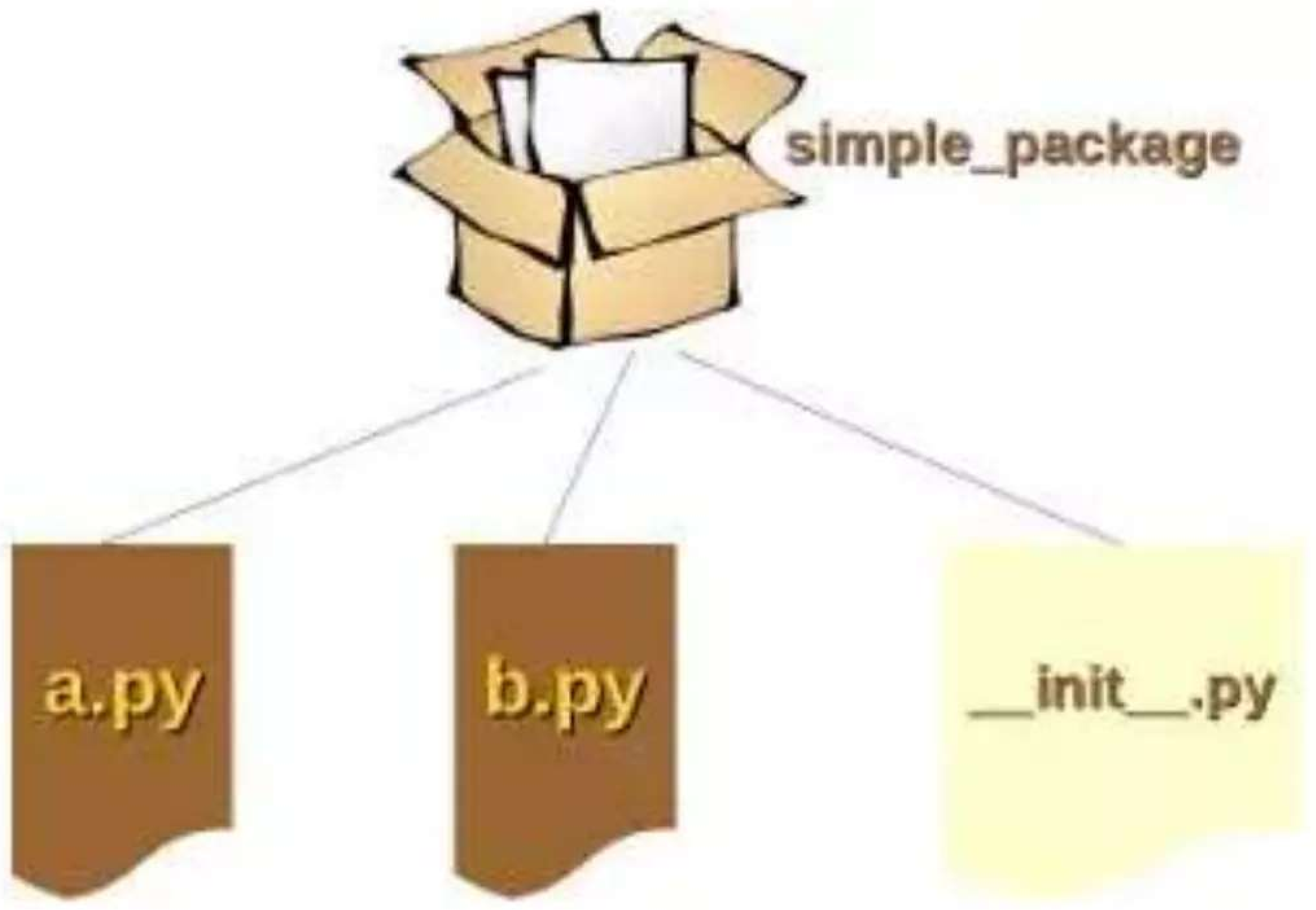
-
  ```
  from mymodule import *
  print(greeting(“Hello”)
  print (person1["age"])
  ```

What are packages?

- We don't usually store all of our files on our computer in the same location.
- We use a **well-organized hierarchy** of directories for easier access.
- Similar **files** are kept in the **same directory**, for example, we may keep **all the songs** in the "**music**" directory.
- similar to this, Python has packages for directories and modules for files

- As our application program grows **larger in size with a lot of modules**, we place **similar modules in one package** and different modules in different packages.
- This makes **a project (program) easy to manage** and **conceptually clear**.
- Similarly, **as a directory can contain subdirectories and files**, a **Python package can have sub-packages and modules**.

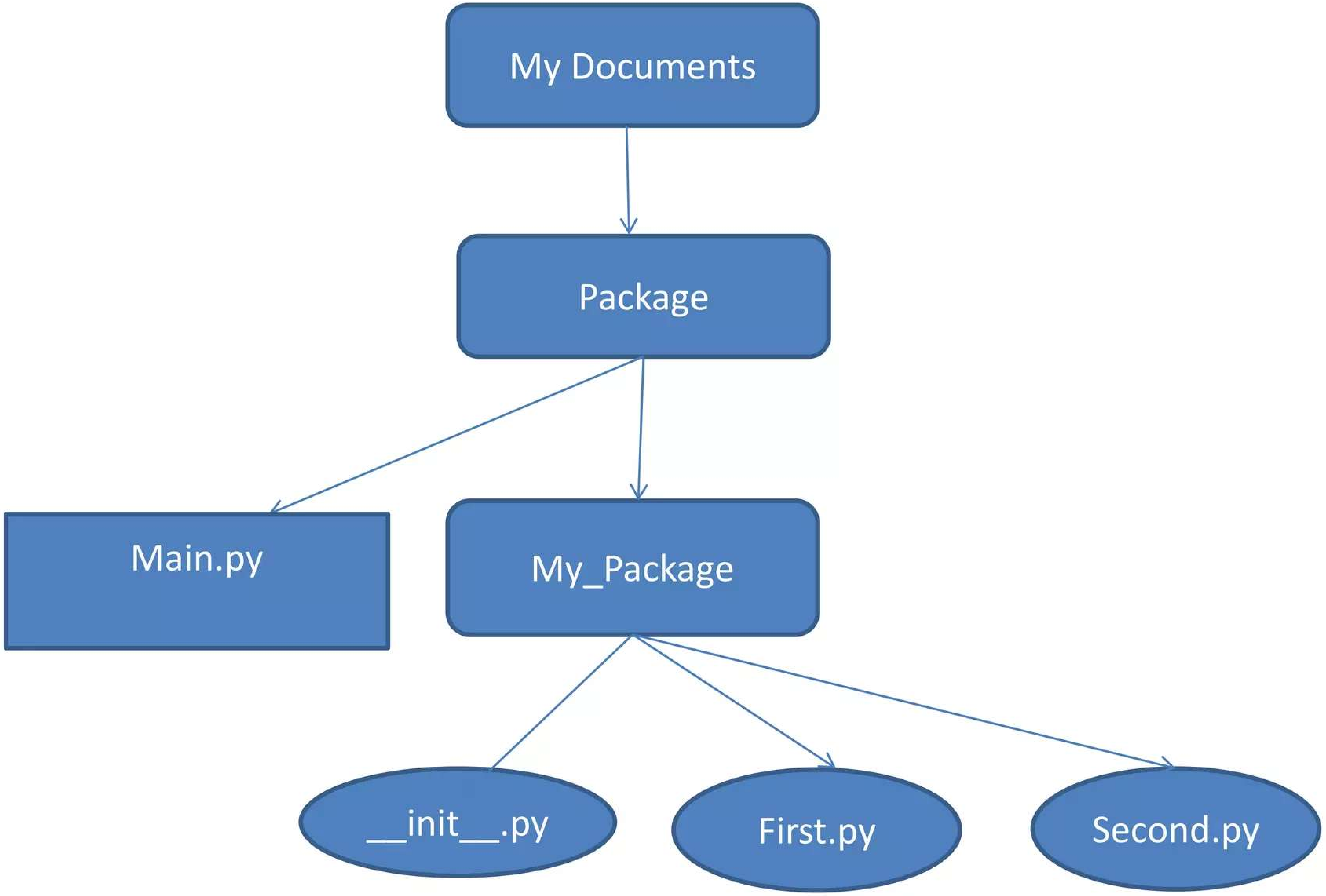
A Simple Example



- First of all, **we need a directory**. The name of this directory will be the name of the package, which we want to create.
- We will **call our package "simple_package". This directory needs to contain a file with the name __init__.py**.
- This file can be empty, or it can contain valid Python code.
- This code will be **executed when a package is imported**, so it can be used to initialize a package.
- We create two simple files a.py and b.py just for the sake of filling the package with modules

__init__.py

- A **directory must contain a file named __init__.py** in order for Python to consider it as a package.
- This file can be left empty but we generally place the initialization code for that package in this file


My Documents
Package
Main.py
My_Package
__init__.py
First.py
Second.py

- **First.py**

```
def one():
    print(“First Module”)
    return
```

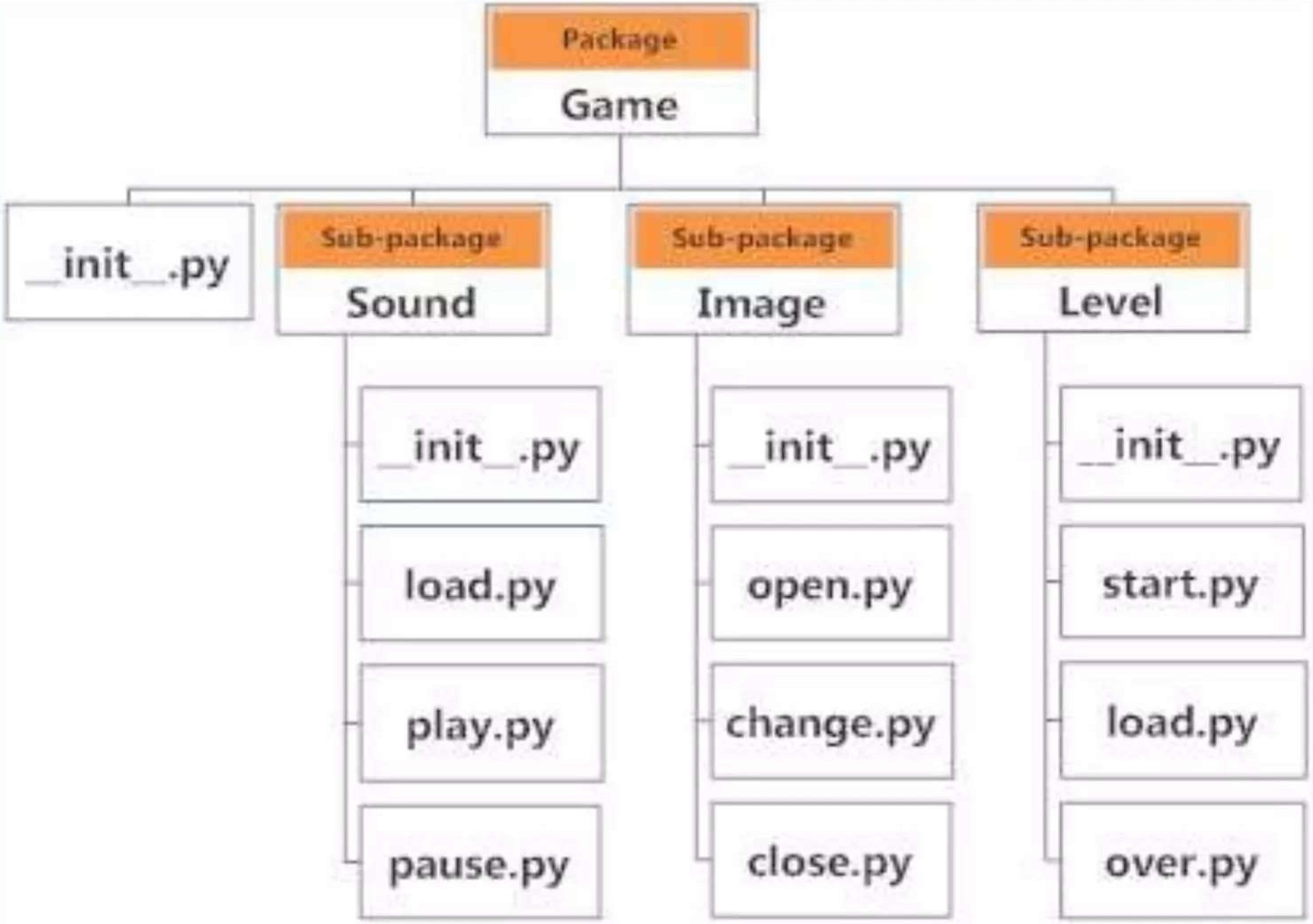
- **Second.py**

```
def second():
    print(“Second Module”)
    return
```

Main.py

- from My-Package import First
- First.one()

- from My-Package import First,Second
- First.one()
- Second.second()



Package Module Structure in Python Programming

- For example, if we want to **import the start module** in the above example, it can be done as follows:
- import Game.Level.start

- Now, if this **module contains a function named select_difficulty()**, we must use the full name to reference it.
- Game.Level.start.select_difficulty(2)

- If this construct seems lengthy, we can **import the module without the package** prefix as follows:
- from Game.Level import start

- We can now call the function simply as follows:
- start.select_difficulty(2)

- Another way of importing just the required function (or class or variable) from a module within a package would be as follows:
- from Game.Level.start import select_difficulty

Now we can directly call this function.

- select_difficulty(2)